SPECIFICATION AMENDMENTS

Please amend the specification as follows:

Substitute paragraph [0001] with the following:

[0001] The present invention generally relates to the field of relational databases and, more specifically, to systems and methods pertaining to an "SQL Algebrizer" (or, more simply, ~~and~~ an "Algebrizer") for transforming SQL syntax tree representations ("SQL ~~Tree~~ Trees") of relational database queries ("SQL Text") into relational algebra representations ("QP Algebra").

Substitute paragraph [0003] with the following:

[0002] Queries are made to a database in the form of SQL Text which, when parsed, is converted into a SQL Tree that must be transformed into QP Algebra for processing by ~~as~~ a SQL Query Processor ("QP"). As known and appreciated by those of skill in the art, the task of transforming a SQL Tree to QP Algebra is performed by an algebrizer.

Substitute paragraph [0004] with the following:

[0003] In general, an algebrizer determines if a relational database query—SQL Text that has been parsed/converted into a SQL Tree—is semantically correct and if so, transforms the SQL Tree into QP Algebra (a specific form of relational algebra understandable to a QP). One approach is for an algebrizer to process a SQL Tree recursively in a depth-first fashion by making one pass per "algebrizing" operation. However, while an algebrizer typically performs more than one distinct operation to "algebrize" a syntax tree representation of a relational database query into a relational algrebra ~~representation , a~~ representation, a typical algebrizer does not typically do any constant folding, which is an operation that is usually performed by the QP (and discussed in more detail later herein).

Substitute paragraph [0007] with the following:

[0004] More specifically, the following steps are performed in a single pass through the SQL Tree for various embodiments of the present invention:

- table and column binding: for every column name in the query, determining which table and column in the database it refers to;
- aggregate binding: for every aggregate function, determining which query specification ~~it~~ should be computed;
- type derivation: for every scalar expression in the query, determining the type of the expression;
- constant folding: if a scalar expression is constant (does not depend on the data in the database), determining the value of the expression (which is normally a step undertaken by the QP, not an algebrizer);
- property derivation: determining whether the statement being processed is deterministic, precise, accesses database data, and other properties of the statement that are necessary for the correct execution of the statement; and
- tree translation: where the resultant in-process SQL Tree node—enhanced and modified by the previous steps—is translated directly into QP Algebra.

Substitute paragraph [0012] with the following:

[0005] Fig. 3B ~~Fig. 3B~~ shows an example of a join of two tables;

Substitute paragraph [0016] with the following:

[0006] Numerous embodiments of the present invention may execute on a computer. Fig. 1 and the following discussion ~~is~~ are intended to provide a brief general description of a suitable computing environment in which the invention may be implemented. Although not required, the invention will be described in the general context of computer executable instructions, such as program modules, being executed by a computer, such as a client workstation or a server. Generally, program modules include routines, programs, objects, components, data structures and the like that perform particular tasks or implement particular abstract data types. Moreover, those skilled in the art will appreciate that the invention may be practiced with other computer system configurations, including hand held devices, multi processor systems, microprocessor based or programmable consumer electronics, network PCs, minicomputers, mainframe computers and the like. The invention may also be practiced in distributed computing environments where tasks are performed by

remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

Substitute paragraph [0021] with the following:

[0007] Many modern database systems, and specifically those based on the relational model, store data in the form of tables. A table is a collection of data organized into rows and columns. Fig. 2 shows an exemplary table 200. In this example, table 200 is a list of bank customers, showing each customer's branch and balance. Table 200 has rows 202 and columns 204. Each column 204 has a name 206. Table 200 may also have a name 208. In the example of Fig. 2, table 200 has the name 208 "Customers." ~~Table 200 has three columns 204.~~ The names 206 of columns 204 are "cust_name," "branch," and "balance," respectively. The first row 202 of table 200 contains the data "James," "Key West," and "$1000." In the terminology of database systems, table 200 is sometimes referred to as a "relation," each row 202 as a "tuple," and the name 206 of each column as an "attribute." It will be appreciated that the depiction of table 200 in Fig. 2 is merely exemplary. A table may have any number of rows

and columns, and may store any type of data, without departing from the spirit and scope of the invention.

Substitute paragraph [0023] with the following:

[0008] Relational algebra comprises a plurality of operations that can be performed on tables and, for example, includes a set of operators that take one or more tables as operands and produce a new table as a result. One important operation in ~~the~~ relational algebra is the "Cartesian product." The Cartesian product is a binary operation that takes two tables as operands and produces a third table as a result. The Cartesian product of two tables R and S (written as R × S) is formed by pairing each row of R with all the rows of S.

Substitute paragraph [0024] with the following:

[0009] Fig. 3A shows an example of the Cartesian product of table 300a and 300b. Table 300a has three columns ("emp_name," "dep't," and "salary"), and table 300b has two columns ("dep't" and "bldg"). The Cartesian product of tables 300a and 300b is a third table 300c having five columns. It will be observed that the five columns of table 300c are the three columns of table 300a plus the two columns of table 300b. (In Fig. 3A, the name of each column in the product table indicates the name of the table from which that column originated. Thus, the first column is named "Employees.emp_name," the second column "Employees.dep't," etc.) ~~Each row~~ Rows of table 300c ~~is~~ are formed by taking a row of table 300a and pairing it with ~~all~~ each of the rows of table 300b. Thus, the first row of table 300c is formed by concatenating the first row of table 300a with the first row of table 300b. The second row of table 300c is formed by concatenating the first row of table 300a with the second row of table 300b. After the rows of table 300b have been exhausted, the next row of table 300a is paired with each row of table 300b to produce the third and fourth rows of table 300c. The process is repeated for each row of table 300a until all the rows of table 300a have been exhausted. It will be appreciated that if table R has R_R rows and R_C columns, and table S has S_R rows and S_C columns, then the Cartesian product R × S is a table having R_R S_R rows and $R_C + S_C$ columns.

Substitute paragraph [0026] with the following:

[0010] Fig. 3B shows an example of a join. Specifically, table ~~200d~~ 300d is the table that results from performing a join on tables ~~200a and 200b~~ 300a and 300b, where the predicate P is "Employees.dep't = Department.dep't." As discussed above, Fig. 3A shows the Cartesian product ~~200c~~ 300c of tables ~~200a and 200b~~ 300a and 300b. Thus, the join of tables ~~200a~~ 300a and ~~200b~~ 300b using the predicate P consists of all of the rows of table ~~200c~~ 300c that meet the condition "Employees.dep't = Dep't.dept." A row meets the predicate P if the value of Employees.dep't for that row is equal to the value of Dep't.dept for that same row. As shown in Fig. 3B, this condition is met by rows 1, 4 and 5 of table ~~200c~~ 300c, and thus table ~~200d~~ 300d consists of those three rows of table ~~200c~~ 300c. Rows 2, 3, and 6 of table ~~200c~~ 300c have different values in the Employees.dep't and Department.dep't columns; thus, rows 2, 3, and 6 do not meet the predicate P and are not included in the result of $R \bowtie_P S$.

Substitute paragraph [0032] with the following:

[0011] Aggregate Binding 418 is a necessary step because, in QP Algebra, an aggregate may only be a child of a GbAgg relational operator, whereas in SQL (e.g., SQL Text) aggregates are freely used in many contexts where a scalar expression is allowed. GbAgg operators, on the other hand, correspond to QuerySpecs in the input tree. Consider the following example:

SELECT c1 FROM t1 GROUP BY c1 HAVING EXISTS

(SELECT * FROM t2 WHERE t2.x > MAX(t1.c2))

In this example, the MAX aggregate is in fact computed in the outer QuerySpec, although it is syntactically located in the inner one. However, in SQL, there is no explicit notion for this—the decision is made implicitly based on the aggregate's argument. Consequently, every aggregate needs to be bound to its hosting QuerySpec in QP Algebra, and the process of doing so is what is referred to herein as "aggregate binding", and is the process undertaken by the step of Aggregate Binding 418.

Substitute paragraph [0033] with the following:

[0012] Type Derivation 420 is the step where the types of any scalar nodes and full metadata for all relational nodes are determined, a necessary step since TSQL is statically typed. (TSQL, a.k.a. "Transact-SQL," is a set of programming extensions that ~~add~~ adds several features to the SQL including transaction control, exception and error handling, row processing, and declared variables.) This step is done in a bottom-up fashion, starting from leaf nodes: columns (whose type information is read from the catalogs) and constants. Then, for non-leaf nodes, the type information is derived based on the particular node type and the types of its children nodes.

Substitute paragraph [0035] with the following:

[0013] To complete each of the aforementioned steps, the typical algebrizer of ~~Fig. XX~~ Fig. 4 requires six passes through the SQL Tree. However, none of these passes include a step of Constant Folding 432 which, as illustrated in Fig. 4, is performed by the QP 430 before undertaking the step of QP Command Processing 434.

Substitute paragraph [0036] with the following:

[0014] Constant Folding 432 is a process whereby queries are simplified by removing statements that inevitably resolve to a scalar value. For example, consider the following SQL statement:

SELECT c1 FROM t WHERE c1 > 3 AND

(SIN(30) * SIN(30) + COS(30) * COS(30) = 1)

In this example, it is clear to see that the subpart of the second condition of the AND statement, "$\sin^2(30) + \cos^2(30) = 1$", always returns a value of "TRUE" because $\sin^2(x) + \cos^2(x) = 1$, and thus this second condition can be reduced to the single value of "TRUE." Further, since the result of an AND statement is "TRUE" if ~~either condition is~~ both conditions are "TRUE," and since the second condition is always "TRUE," the AND statement ~~itself is also always "TRUE."~~ can be removed. Thus, the original example statement can be rewritten as follows:

SELECT c1 FROM t WHERE c1 > 3

In this regard, Constant Folding 432 is the process that identifies and replaces ("folds") these kinds of scalar values ("constants").

Substitute paragraph [0038] with the following:

[0015] Fig. 5 is a block diagram illustrating one embodiment of the present invention for efficiently and more completely transforming SQL Text into input for the QP. SQL Text 502 is inputted into a Parser 504 that converts said SQL Text 502 into a SQL Tree 506 as output. This SQL Tree 506 is then inputted into the SQL Algebrizer 510. Then, in a single "Algebrizing" pass 514, the Algebrizer 510 performs the following operations sequentially at each node of the SQL Tree 506: Table and Column Combining 516; Aggregate Binding 518; Type Derivation 520; Constant Folding 232 (heretofore not performed in the Algebrizer); Property Derivation 522; and Tree Translation 524. The net result of this single Algebrizer Pass ~~5214~~ 514 is Optimized QP Algebra 528 (optimized in the sense that this QP Algebra has already undergone the step of Constant Folding ~~5232)~~ 532), which is then inputted into the Query Processor 530 for immediate processing by the QP Command Process 534.

Substitute paragraph [0041] with the following:

[0016] While the present invention has been described in connection with the preferred embodiments of the various figures, it is to be understood that other similar embodiments may be used or modifications and additions may

be made to the described embodiment for performing the same function of the present invention without deviating there from. For example, while exemplary embodiments of the invention are described in the context of digital devices emulating the functionality of personal computers, one skilled in the art will recognize that the present invention is not limited to such digital devices. As ~~, as~~ described in the present application, exemplary embodiments may be implemented using ~~may apply to~~ any number of existing or emerging computing devices or environments, such as a gaming console, handheld computer, portable computer, etc. whether wired or wireless, and may be ~~applied to~~ implemented using any number of such computing devices connected via a communications network, and interacting across the network. Furthermore, it should be emphasized that a variety of computer platforms, including handheld device operating systems and other application specific hardware/software interface systems, are herein contemplated, especially as the number of wireless networked devices continues to proliferate. Therefore, the present invention should not be limited to any single embodiment, but rather construed in breadth and scope in accordance with the appended claims.